Overlay Networks, Decentralized Systems and their Application

Exercise 03

University of Zurich

Reto WETTSTEIN
12-716-221
reto.wettstein2@uzh.ch

Solved together with Christian TRESCH 06-923-627

March 10, 2015

1 Bloom Filters

1 **Which operations does the traditional Bloom Filter support?**
INSERTION: The bit A[h_i(x)] for $1 < i < k$ are set to 1, where k is the number of hash. functions.
QUERY: Yes if all of the bits A[h_i(x)] are 1, no otherwise.
DELETION: Removing an element from a simple Bloom filter is impossible.
UNION: Bitwise OR of Bloom Filters.
INTERSECTION: Bitwise AND of Bloom Filters.

2 **Does the Bloom Filter have a capacity limit? What changes if more and more elements are added?**
The Bloom Filter can represent the entire universe of elements (in this case all bits are 1) → no capacity limit exists, an INSERTION always works. If more and more elements are added, the rate of false-positive answers increases steadily.

3 **What is a false positive? Explain how it can happen.**
A false-positive occurs, when the result of a query is yes, even if the element is not in the set. For example the insertion of hash('Computer') sets bits (1, 5, 7) to one and the insertion of hash('Science') sets bits (2, 4, 6) to one. Then the query hash('Machine') = (1, 2, 6) returns yes although 'machine' was never inserted into the filter.

4 **Can the traditional Bloom Filter have false negatives? Explain.**
No, the traditional Bloom Filter can have no false negatives. If the corresponding bits are set to one and match the query, the Bloom Filter always returns yes. Bits set to one can never change back to zero, because deletion is not possible.

5 **Describe a real life application scenario for Bloom Filters and explain why they are useful in the chosen scenario?**
SPELL-CHECKING: a Bloom filter is used to store a dictionary of correctly-spelled words. If hash('word') returns false, the spell checker flags word as a misspelling. False positives in this application, e.g., hash('notaword') == True, results in some misspellings going unnoticed. Bloom filters allow the spell checking application to load a comprehensive dictionary into a small amount of memory and makes spell checking fast enough that users can run the checker often. The small memory footprint can be achieved with a false positive rate that results in approximately 1 in 100 misspellings going undetected.

2 Kademlia

1 **How many ID's are possible?**
A node ID has a length of 160 bit → $2^{160} \approx 1.5 \text{ x } 10^{48}$ different ID's are possible.

2 **Where is a key located?**
A key is located on the node whose ID is closest to the key.

3 **What is the XOR distance betweeen 3 and 4?**
7 → 011_2 XOR $100_2 = 111_2$, which is 7_{10}

4 **Kademlia routing tables consist of a list for each bit of the node ID. (e.g. if a node ID consists of 128 bits, a node will keep 128 such lists. In this case, would 127 lists be enough? Why?**
Yes, 127 lists would be enough because you don't need to know yourself.

3 Challenge Task Preparation

This is my code for the programming task. The output looks like this:

```
PEER 3: stored [Key: Max Power, Value: paddr[0x3[/192.168.0.17,4001]]/relay(false)/slow(false)]

PEER 5: looked up [Key: Max Power], received [Value: paddr[0x3[/192.168.0.17,4001]]/relay(false)/slow(false)]

PEER 3: received [Message: Hello World] from peer 5
```

```java
package net.tomp2p.exercise.retowettstein.ex03;

import java.io.IOException;

import net.tomp2p.dht.FutureGet;
import net.tomp2p.dht.FuturePut;
import net.tomp2p.dht.PeerDHT;
import net.tomp2p.peers.PeerAddress;


/**
 * @author Reto Wettstein 12-716-221
 * @author Christian Tresch 06-923-627
 */
public class Main {

    public static final int NUMBER_OF_PEERS   = 10;
    public static final int STORING_PEER_INDEX = 2; // peerIndex is 1 smaller than peerId
    public static final int GETTER_PEER_INDEX = 4; // peerIndex is 1 smaller than peerId
    public static final String KEY            = "Max Power";
    public static final int PORT              = 4001;

    public static void main(String[] args) {
        PeerDHT[] peers = null;

        try {
            peers = DHTOperations.createAndAttachPeersDHT(NUMBER_OF_PEERS, PORT);
            DHTOperations.bootstrap(peers);
            SendOperations.setupReplyHandler(peers);

            PeerAddress value = peers[STORING_PEER_INDEX].peerAddress();
            String message = "Hello World";

            FuturePut futurePut = DHTOperations.putNonBlocking(peers[STORING_PEER_INDEX], KEY, value);
            futurePut.await();

            FutureGet futureGet = DHTOperations.getNonBlocking(peers[GETTER_PEER_INDEX], KEY);
            futureGet.await();

            PeerAddress address = (PeerAddress) futureGet.data().object();
            SendOperations.send(peers[GETTER_PEER_INDEX], address, message);

            Thread.sleep(1000);

            DHTOperations.peersShutdown(peers);
        } catch (IOException pEx) {
            pEx.printStackTrace();
        } catch (InterruptedException pEx) {
            pEx.printStackTrace();
        } catch (ClassNotFoundException pEx) {
            pEx.printStackTrace();
        }
    }

}
```

```java
package net.tomp2p.exercise.retowettstein.ex03;

import java.io.IOException;
import net.tomp2p.dht.FutureGet;
import net.tomp2p.dht.FuturePut;
import net.tomp2p.dht.PeerBuilderDHT;
import net.tomp2p.dht.PeerDHT;
import net.tomp2p.futures.BaseFutureAdapter;
import net.tomp2p.p2p.PeerBuilder;
import net.tomp2p.peers.Number160;
import net.tomp2p.peers.PeerAddress;
import net.tomp2p.storage.Data;


public class DHTOperations {

    /**
     * Create peers with a port and attach it to the first peer in the array.
     *
     * @param nr The number of peers to be created
     * @param port The port that all the peer listens to. The multiplexing is done via the peer Id
     * @return The created peers
     * @throws IOException IOException
     */
    public static PeerDHT[] createAndAttachPeersDHT(int nr, int port)
            throws IOException {
        PeerDHT[] peers = new PeerDHT[nr];
        for (int i = 0; i < nr; i++) {
            if (i == 0) {
                peers[0] = new PeerBuilderDHT(new PeerBuilder(new Number160(i +
                     1)).ports(port).start()).start();
            } else {
                peers[i] = new PeerBuilderDHT(new PeerBuilder(new Number160(i +
                     1)).masterPeer(peers[0].peer()).start()).start();
            }
        }

        return peers;
    }


    /**
     * Bootstraps peers to the first peer in the array.
     *
     * @param peers The peers that should be bootstrapped
     */
    public static void bootstrap(PeerDHT[] peers) {
        // make perfect bootstrap, the regular can take a while
        for (int i = 0; i < peers.length; i++) {
            for (int j = 0; j < peers.length; j++) {
                peers[i].peerBean().peerMap().peerFound(peers[j].peerAddress(), null, null,
                     null);
            }
        }
    }


    /**
     * Put data into the DHT in an asynchronous way.
     *
     * @param pPeer The storing peer
     * @param pKey The key for storing the data
     * @return pValue The address where to find the data
     * @throws IOException IOException
     */
    public static FuturePut putNonBlocking(PeerDHT pPeer, String pKey, PeerAddress pValue)
            throws IOException {
        FuturePut futurePut = pPeer.put(Number160.createHash(pKey)).data(new
             Data(pValue)).start();

        // non-blocking operation
        futurePut.addListener(new BaseFutureAdapter<FuturePut>() {

            @Override
            public void operationComplete(FuturePut future)
                    throws Exception {
                if (future.isSuccess()) {
                    System.out.println("PEER " + pPeer.peerAddress().peerId().intValue() +
                        ": stored " + "[Key: " + pKey + ", Value: " + pValue + "]");
                }
            }
        });

        return futurePut;
    }


    /**
     * Get the address of the peer storing data and send a message
     * to the storing peer in asynchronous way.
     *
     * @param pPeer The peer who does the lookup
     * @param pKey The key corresponding to the data
     * @param pMessage THe message to send to the received address
     */
    public static FutureGet getNonBlocking(PeerDHT pPeer, String pKey) {
        FutureGet futureGet = pPeer.get(Number160.createHash(pKey)).start();

        // non-blocking operation
        futureGet.addListener(new BaseFutureAdapter<FutureGet>() {

            @Override
            public void operationComplete(FutureGet future)
                    throws Exception {
                if (future.isSuccess()) {
                    PeerAddress address = (PeerAddress) future.data().object();
                    System.out.println("PEER " + pPeer.peerAddress().peerId().intValue() +
                        ": looked up [Key: " + pKey + "], received [Value: " + address +
                        "]");
                }
            }
        });

        return futureGet;
    }


    /**
     * Shutdown peers.
     *
     * @param pPeers The peers that should be shutdown
     */
    public static void peersShutdown(PeerDHT[] pPeers) {
        for (int i = 0; i < pPeers.length; i++) {
            pPeers[i].shutdown();
        }
    }
}
```

```
package net.tomp2p.exercise.retowettstein.ex03;

```

```
import net.tomp2p.dht.FutureSend;
import net.tomp2p.dht.PeerDHT;
import net.tomp2p.futures.BaseFutureAdapter;
import net.tomp2p.p2p.RequestP2PConfiguration;
import net.tomp2p.peers.PeerAddress;
import net.tomp2p.rpc.ObjectDataReply;


public class SendOperations {

    /**
     * Setup a reply handler for every peer in the network
     *
     * @param peers Array with the peers who need a reply handler
     */
    public static void setupReplyHandler(PeerDHT[] peers) {
        for (final PeerDHT peer : peers) {
            peer.peer().objectDataReply(new ObjectDataReply() {

                @Override
                public Object reply(PeerAddress sender, Object request)
                        throws Exception {
                    System.out.println("PEER " + peer.peerID().intValue() + ": received
                         [Message: " + request + "] from peer " +
                         sender.peerId().intValue());
                    return "world";
                }
            });
        }
    }


    /**
     * Send a direct message from one peer to another
     *
     * @param sender The peer sending the message
     * @param receiver The peer address of the receiving peer
     * @param message The message to be sent
     */
    public static void send(PeerDHT sender, PeerAddress receiver, String message) {
        RequestP2PConfiguration requestP2PConfiguration = new RequestP2PConfiguration(1,
            10, 0);
        FutureSend futureSend =
            sender.send(receiver.peerId()).object(message).requestP2PConfiguration(requestP2PConfiguration).sta

        // non-blocking operation
        futureSend.addListener(new BaseFutureAdapter<FutureSend>() {

            @Override
            public void operationComplete(FutureSend future)
                    throws Exception {
                if (!future.isSuccess()) {
                   // Some error message
                }
            }
        });
    }
}
```